



# UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE  
United States Patent and Trademark Office  
Address: COMMISSIONER FOR PATENTS  
P.O. Box 1450  
Alexandria, Virginia 22313-1450  
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/480,309	01/10/2000	DAVID N. WILNER	11283/2	4170

7590 03/10/2004

KENYON & KENYON  
333 W SAN CARLOS STREET  
SUITE 600  
SAN JOSE, CA 95110-2711

EXAMINER
----------

ALI, SYED J

ART UNIT	PAPER NUMBER
----------	--------------

2127

DATE MAILED: 03/10/2004

13

Please find below and/or attached an Office communication concerning this application or proceeding.

**Office Action Summary**

Application No.

09/480,309

Applicant(s)

WILNER ET AL.

Examiner

Syed J Ali

Art Unit

2127

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --  
**Period for Reply**

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

**Status**

- 1) ☒ Responsive to communication(s) filed on December 15, 2003.
- 2a) ☒ This action is **FINAL**. 2b) ☐ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

**Disposition of Claims**

- 4) ☒ Claim(s) 1-34 is/are pending in the application.
- 4a) Of the above claim(s) \_\_\_\_\_ is/are withdrawn from consideration.
- 5) ☐ Claim(s) \_\_\_\_\_ is/are allowed.
- 6) ☒ Claim(s) 1-34 is/are rejected.
- 7) ☐ Claim(s) \_\_\_\_\_ is/are objected to.
- 8) ☐ Claim(s) \_\_\_\_\_ are subject to restriction and/or election requirement.

**Application Papers**

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☐ The drawing(s) filed on \_\_\_\_\_ is/are: a) ☐ accepted or b) ☐ objected to by the Examiner.  
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).  
Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

**Priority under 35 U.S.C. § 119**

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some \* c) ☐ None of:
- ☐ Certified copies of the priority documents have been received.
  - ☐ Certified copies of the priority documents have been received in Application No. \_\_\_\_\_.
  - ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

\* See the attached detailed Office action for a list of the certified copies not received.

**Attachment(s)**

- ☐ Notice of References Cited (PTO-892)
- ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08)  
Paper No(s)/Mail Date \_\_\_\_\_
- ☐ Interview Summary (PTO-413)  
Paper No(s)/Mail Date. \_\_\_\_\_
- ☐ Notice of Informal Patent Application (PTO-152)
- ☐ Other: \_\_\_\_\_

### DETAILED ACTION

1. This office action is in response to Amendment A, paper number 12, which was filed December 15, 2003. Claims 1-34 are presented for examination.
2. The text of those sections of Title 35, U.S. code not included in this office action can be found in a prior office action.

### *Claim Rejections - 35 USC § 102*

3. Claims 1-3, 8-10, 15-19, and 21-23 are rejected under 35 U.S.C. 102(b) as being anticipated by Kempf et al. (previously cited) (hereinafter Kempf).

As per claim 1, Kempf teaches the invention as claimed, including a method, comprising the steps of:

loading a code module into a memory space of a first domain (col. 6 line 60 - col. 7 line 22, "During a mapping operation, the program code and/or the group of data is inserted into the address space in bulk, and the addresses of the address space is associated with the program code and/or data"), the code module including an instruction having a symbol reference (col. 6 line 60 - col. 7 line 22, "Additionally, during a mapping operation, symbol address information about a program segment is obtained from the program segment by accessing the locations where the program segment is deposited");

determining if the symbol reference is to an external location outside of the memory space (col. 9 lines 3-13, "If the code table indicates that the program code segment has not been

Art Unit: 2127

linked in the address space, the client then obtains the program code segment object from a program code manager”);

generating a link stub for the symbol reference when the symbol reference is to an external location to access the external location (col. 9 lines 14-25, “The client process then maps the program code segment into the address space and into its own address space and links the program code segment relative to addresses in the executing process”); and

redirecting the instruction to the link stub (col. 9 lines 14-25, “Upon linking the program code segment to the executing process, the client process then locates the initialization function of the program code segment, block 100. Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function”).

As per claim 2, Kempf teaches the invention as claimed, including the method of claim 1, wherein the link stub is part of a linking table entry corresponding to the symbol reference (col. 7 lines 23-44, “The code table comprises entries for each linked program code segment for the address space”).

As per claim 3, Kempf teaches the invention as claimed, including the method of claim 1, wherein the link stub is a jump instruction to the external location (col. 9 lines 14-25, “Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function”).

As per claim 8, Kempf teaches the invention as claimed, including a method, comprising:  
creating a task in a first domain, the task executing a number of instructions (col. 6 line 60 - col. 7 line 22, "During a mapping operation, the program code and/or the group of data is inserted into the address space in bulk, and the addresses of the address space is associated with the program code and/or data");

executing a first jump instruction in the number of instructions that refers to a link stub corresponding to an external location in a second domain (col. 9 lines 14-25, "Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function");

executing the link stub (col. 9 lines 14-25, "Upon linking the program code segment to the executing process, the client process then locates the initialization function of the program code segment, block 100. Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function").

As per claim 9, Kempf teaches the invention as claimed, including the method of claim 8, wherein the link stub is part of linking table entry corresponding to the external location (col. 7 lines 23-44, "The code table comprises entries for each linked program code segment for the address space").

Art Unit: 2127

As per claim 10, Kempf teaches the invention as claimed, including the method of claim 8, wherein the link stub includes a second jump instruction to the external location (col. 9 lines 14-25, "Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function").

As per claim 15, Kempf teaches the invention as claimed, including a computer system, comprising:

a system space having a number of memory locations (col. 6 line 60 - col. 7 line 9, "The address space object 62 comprises an address space [not shown] and provides an object oriented interface for performing limited operations on the address space on behalf of client processes executing in non-supervisor mode. The limited operations comprise an operation for returning a list of memory objects mapped into the address space and the corresponding base addresses for the memory objects");

a number of protection domains, at least one of the number of protection domains owning a portion of the system space (col. 6 line 60 - col. 7 line 9, "The limited operations comprise...an operation of mapping a program code segment and/or a group of data into the address space, producing a memory object corresponding to the mapped memory").

As per claim 16, Kempf teaches the invention as claimed, including the computer system of claim 15, wherein the at least one of the number of protection domains includes at least one of:

Art Unit: 2127

a code module (col. 6 line 60 - col. 7 line 9, “During a mapping operation, the program code and/or the group of data is inserted into the address space in bulk, and the addresses of the address space is associated with the program code and/or data”);

a link stub (col. 9 lines 14-25, “The client process then maps the program code segment into the address space and into its own address space and links the program code segment relative to addresses in the executing process”); and

an entry point (col. 9 lines 14-25, “Upon linking the program code segment to the executing process, the client process then locates the initialization function of the program code segment, block 100. Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function”).

As per claim 17, Kempf teaches the invention as claimed, including the system of claim 16, wherein the link stub is part of a linking table entry in a linking table (col. 7 lines 23-44, “The code table comprises entries for each linked program code segment for the address space”).

As per claim 18, Kempf teaches the invention as claimed, including the system of claim 16, wherein the entry point is represented in a symbol table (col. 9 lines 14-25, “Upon linking the program code segment to the executing process, the client process then locates the initialization function of the program code segment, block 100”).

Art Unit: 2127

As per claim 19, Kempf teaches the invention as claimed, including 19 the system of claim 16, wherein at least one of the number of protection domains is a system protection domain that includes:

at least one code module including executable code for operating system services (col. 5 lines 36-54, "The system software 30 comprises an operating system"); and

at least one system object owned by the system protection domain (col. 5 lines 36-54, "Operating system 32 provides...an object oriented interface for securely mapping files into address spaces on behalf of a process executing in non-supervisor mode").

As per claim 21, Kempf teaches the invention as claimed, including the system of claim 19, wherein at least one of the number of protection domains is a first protection domain that includes:

at least one code module including executable code for a first set of functions (col. 6 line 60 - col. 7 line 9, "During a mapping operation, the program code and/or the group of data is inserted into the address space in bulk, and the addresses of the address space is associated with the program code and/or data"); and

a number of link stubs (col. 9 lines 14-25, "The client process then maps the program code segment into the address space and into its own address space and links the program code segment relative to addresses in the executing process");

wherein at least one of the link stubs corresponds to a symbol referenced in the executable code for the first set of functions (col. 6 line 60 - col. 7 line 22, "Additionally, during a mapping operation, symbol address information about a program segment is obtained from the



Art Unit: 2127

program segment by accessing the locations where the program segment is deposited”), and such at least one link stub includes executable code to direct execution to the executable code for operating system services (col. 9 lines 14-25, “Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function”).

As per claim 22, Kempf teaches the invention as claimed, including the system of claim 21, wherein the number of protection domains includes a second protection domain that includes:

at least one code module including executable code for a second set of functions (col. 9 lines 14-25, “The client process then maps the program code segment into the address space and into its own address space and links the program code segment”, wherein the program code segment being linked is in another namespace, or protection domain); and

a number of entry points (col. 9 lines 14-25, “the client process then locates the initialization function of the program code segment”, wherein the initialization function is the entry point);

wherein each of the entry points corresponds to a symbol in the executable code for the second set of functions (col. 9 lines 14-25, “Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function”).

As per claim 23, Kempf teaches the invention as claimed, including the system of claim 22, wherein one of the link stubs of the first protection domain corresponds to one of the number

Art Unit: 2127

of entry points in the second protection domain, and such link stub includes executable code to direct execution to the one of the number of entry points in the second protection domain (col. 9 lines 14-25, "Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function").

4. Claims 28-29 and 32-34 are rejected under 35 U.S.C. 102(e) as being anticipated by Van Doorn (previously cited).

As per claim 28, Van Doorn teaches the invention as claimed, including a protection domain, comprising:

a memory space (col. 4 lines 55-62, "Memory management is separated into physical and virtual memory management", wherein the memory space is mapped to different protection domains and all management of this memory space is done by the Java Virtual Machine); and

a protection view designating a set of protection domains for unrestricted memory access (col. 5 lines 17-26, "Each protection domain has a view of its own subtree of the name space, the kernel address space has a view of the entire tree including all the subtrees of different protection domains").

As per claim 29, Van Doorn teaches the invention as claimed, including the protection domain of claim 28, further comprising:

Art Unit: 2127

a number of code modules, each of the number of code modules including at least one of executable code and data structures (col. 3 lines 48-65, "The present invention is most advantageously applied to the integration of a programming language system and an operating system when the programming language exhibits the following system properties: 1. a well defined unit of protection embedded in the language. These units form the basis of the protection. Examples of these units are classes, objects, and modules").

As per claim 32, Van Doorn teaches the invention as claimed, including the protection domain of claim 28, further comprising:

authorization information for enabling attachment of other protection domains (col. 8 line 63 - col. 9 line 6, "the memory available to all protection domains is mapped into the Java Nucleus with read/write permission. This allows it to quickly access the data in different protection domains").

As per claim 33, Van Doorn teaches the invention as claimed, including the protection domain of claim 28, further comprising:

a task, having a task control block (this is handled by the Java runtime context), a task protection view (col. 5 lines 17-27, "Each protection domain has a view of its own subtree of the name space, the kernel address space has a view of the entire tree including all the subtrees of different protection domains"), a protection switch stack (col. 4 lines 36-54, "A handler consists of a protection domain identifier, the address of a call-back function, and a stack pointer", wherein the stack pointer identifies all the namespaces within the protection view of the

Art Unit: 2127

protection domain, and is located through the use of the protection domain identifier) and a task privilege level (col. 8 line 63 - col. 9 line 6, “the memory available to all protection domains is mapped into the Java Nucleus with read/write permission. This allows it to quickly access the data in different protection domains”).

As per claim 34, Van Doorn teaches the invention as claimed, including the protection domain of claim 28, further comprising:

protection domain attributes, including an indication whether the protection domain may be attached (col. 8 line 63 - col. 9 line 6, “the memory available to all protection domains is mapped into the Java Nucleus with read/write permission. This allows it to quickly access the data in different protection domains”).

### ***Claim Rejections - 35 USC § 103***

5. Claims 4-7, 11, 20, 24-27, and 30-31 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kempf in view of Van Doorn.

As per claim 4, Van Doorn teaches the invention as claimed, including the following limitations not shown by Kempf, specifically the method of claim 1, further comprising the steps of:

determining if the external location is within a second domain that is within a protection view of the first domain (col. 5 lines 17-27, “Each protection domain has a view of its own

Art Unit: 2127

subtree of the name space, the kernel address space has a view of the entire tree including all the subtrees of different protection domains”);

requesting attachment of the second domain to the first domain when the second domain is determined not to be within the protection view of the first domain (col. 8 line 63 - col. 9 line 6, “the memory available to all protection domains is mapped into the Java Nucleus with read/write permissions. This allows it to quickly access the data in different protection domains”, wherein when a first domain requires access to methods of a second domain, the Java Nucleus performs the mapping, or attachment, process);

attaching the second domain to the first domain using an attachment mechanism (col. 9 line 7-23, “memory Region 0 is mapped into the executable content context 330. Part of this memory contains executable code and has the execute privilege associated with it”).

It would have been obvious to one of ordinary skill in the art to add Van Doorn to Kempf since under certain circumstances, a code module may need to access data that is outside of its particular protection domain. In such cases, security measures typically do not allow such access since it may cause a breach. This is a deficiency of Kempf. Van Doorn makes up for this deficiency by allowing a protection domain to add other existing domains to its data protection set, thereby increasing the range of the code, while maintaining security measures. Furthermore, Van Doorn teaches denying access to the second domain when it is not mapped into the context of the first domain, thereby allowing a secure execution environment.

As per claim 5, Van Doorn teaches the invention as claimed, including the method of claim 4, further comprising the steps of:

determining whether the attachment request is permitted based on authorization information provided by the first domain (col. 8 line 63 - col. 9 line 6, “the memory available to all protection domains is mapped into the Java Nucleus with read/write permission. This allows it to quickly access the data in different protection domains”);

wherein attachment to the second domain is not permitted when the attachment request is not permitted (col. 8 line 63 - col. 9 line 6, “the memory available to all protection domains is mapped into the Java Nucleus with read/write permission. This allows it to quickly access the data in different protection domains”, wherein if the memory is not mapped to the Java Nucleus, attachment is not allowed since it may cause a security breach or a page fault).

As per claim 6, Van Doorn teaches the invention as claimed, including the method of claim 4, wherein the attachment mechanism comprises adding the second domain to the protection view of the first domain (col. 9 line 7-23, “memory Region 0 is mapped into the executable content context 330. Part of this memory contains executable code and has the execute privilege associated with it”) and Kempf teaches the invention as claimed, including a jump instruction to the external location in the link stub (col. 9 lines 14-25, “Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function”).

As per claim 7, Kempf teaches the invention as claimed, including the method of claim 4, wherein the attachment mechanism comprises including a jump instruction to the external

Art Unit: 2127

location in the link stub without altering the protection view of the first domain (col. 9 lines 14-25, "Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function", wherein nothing is altered in the protection view of the first domain; rather, control is transferred to the location in memory of the desired instruction).

As per claim 11, Van Doorn teaches the invention as claimed, including the following limitations not shown by Kempf, specifically, the method of claim 8, further comprising the steps of:

comparing the external location to a task protection view (col. 9 lines 41-60, "Consider the protection Domains A and B and a Method M which resides in Domain-B", wherein a determination is made as to whether or not the method called is within either the protection domain or the protection view from the domain that it is called from); and

generating a processing exception when the external location is outside the task protection view (col. 9 lines 41-60, "An instruction fault occurs when A calls Method M, since M is not mapped into context A").

As per claim 20, Van Doorn teaches the invention as claimed, including the system of claim 19, wherein the system protection domain includes a protection domain list that includes entries for each of the number of protection domains (col. 5 lines 17-27, "Each protection domain has a view of its own subtree of the name space, the kernel address space has a view of the entire tree including all the subtrees of different protection domains").

As per claim 24, Van Doorn teaches the invention as claimed, including the system of claim 16, wherein each of the number of protection domains includes a protection view defining a set of the number of protection domains to which unprotected access may be made (col. 5 lines 17-27, "Each protection domain has a view of its own subtree of the name space, the kernel address space has a view of the entire tree including all the subtrees of different protection domains").

As per claim 25, Van Doorn teaches the invention as claimed, including the system of claim 24, wherein the protection view sets a memory range of allowable memory accesses, and wherein a memory fault is generated when memory access is attempted outside of the memory range (col. 5 lines 1-10, "The protection domain's virtual memory space is managed by the virtual memory service", "Each virtual page has associated with it a fault event that is raised, if set, when a fault occurs on an address within that page").

As per claim 26, Van Doorn teaches the invention as claimed, including the system of claim 25, wherein the memory range is contiguous (Fig. 3, elements 310, 320, and 330, wherein the virtual memory pages being mapped are contiguous portions of memory).

As per claim 27, Van Doorn teaches the invention as claimed, including the system of claim 25, further comprising an exception handling routine that is executed on the occurrence of the memory fault, the exception handling routine including a protection switch mechanism (col.



Art Unit: 2127

5 lines 11-16, "The event handlers for a virtual page fault do not have to reside in the same protection domain where the fault occurs", wherein the exception handler executes by finding the requested memory addresses from memory, thereby switching the protection view from only looking at the present protection domain to all memory domains that the Java Nucleus has access to)

As per claim 30, Kempf shows the following limitations not shown by Van Doorn, specifically the protection domain of claim 29, further comprising:

a symbol table comprising a number of symbol entries, at least one of the symbol entries specifying an entry point address within the memory space (col. 3 lines 14-22, "the address space manager is also used to provide a code table for identifying the first program code segment as not being linked in the second process, and to provide symbol address information for generating a symbol address table", wherein the generating of a symbol address table comprises finding the entry point, i.e., initialization function, for that code segment).

As per claim 31, Kempf teaches the invention as claimed, including the protection domain of claim 29, further comprising:

a linking table having a number of linking table entries, each linking table entry including a link stub directed to an address outside of the memory space (col. 7 lines 23-44, "The code table comprises entries for each linked program code segment for the address space").

Art Unit: 2127

6. Claims 12-14 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kempf in view of Kelly et al. (previously cited) (hereinafter Kelly) in view of Van Doorn.

As per claim 12, Kelly teaches the invention as claimed, including the method of claim 11, further comprising the steps of:

executing an exception handling routine in response to the generation of the processing exception, the exception handling routine including:

saving a pre-exception setting of the task protection view (col. 13 lines 39-61, "The use of such an exception handler requires that the code morphing software include routines for emulating the entire exception handling process including any hardware provided by the target computer for handling the process. This requires that the code morphing software provide for saving the state of the target processor so that it may proceed correctly after the exception has been handled").

Van Doorn teaches the invention as claimed, including altering the task protection view to include a protection view of the second domain (col. 9 line 7-23, "memory Region 0 is mapped into the executable content context 330. Part of this memory contains executable code and has the execute privilege associated with it"); and

Kempf teaches the invention as claimed, including jumping to the external location (col. 9 lines 14-25, "Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function").

It would have been obvious to one of ordinary skill in the art to combine Kempf, Van Doorn, and Kelly since saving the state of a computer prior to handling an exception allows the system to protect a known safe state. Therefore, if any error occurred during the exception handling, the system could simply roll back to the saved state. This provides an added security measure for cases when methods outside of a protection view are accessed.

As per claim 13, Van Doorn teaches the invention as claimed, including the method of claim 12, wherein the task protection view is saved on a task protection switch stack (col. 4 lines 36-54, "A handler consists of a protection domain identifier, the address of a call-back function, and a stack pointer", wherein the stack pointer identifies all the namespaces within the protection view of the protection domain, and is located through the use of the protection domain identifier).

As per claim 14, Kelly teaches the invention as claimed, including the method of claim 12, further comprising the steps of:

retrieving the pre-exception setting of the task protection view (col. 13 line 62 - col. 14 line 5, "In these cases, the code morphing software, using the state saving and restoring mechanisms described above, causes the target state to be restored to its most recent official version");

restoring the task protection view using the pre-exception setting of the task protection view (col. 13 line 62 - col. 14 line 5, "In these cases, the code morphing software, using the state

saving and restoring mechanisms described above, causes the target state to be restored to its most recent official version”).

Van Doorn teaches the invention as claimed, including returning to a subsequent instruction to the first jump instruction in the number of instructions (this is inherent in the disclosure of Van Doorn since it is in reference to a Java Virtual Machine that executes bytecode sequentially).

### ***Response to Arguments***

7. Applicant's arguments filed December 15, 2003 have been fully considered but they are not persuasive.

8. Applicant argues on page 10, *“The Kempf reference provides no discussion of generating a link stub, or of redirecting an instruction to a link stub. At best, the Kempf reference describes the well known process of directly linking program code and data through use of a symbol table...[and] transferring control to the linked program at its starting address (or initialization function),...not redirecting an instruction to a link stub.”*

Examiner respectfully disagrees. While Kempf may not use the specific language of “link stub”, the dynamic linking of program code across address spaces achieves exactly the same result as the claimed “protection domain” and “link stub”. Specifically, it is well known that a protection domain seeks to control memory access for all threads and processes within a single address space by ensuring that only threads and processes within that protection domain may have unlimited access to the resources and memory pages within that domain. Furthermore,

Art Unit: 2127

Kempf discloses that when a thread makes a call to a portion of memory that is outside the address space, i.e. protection domain, the appropriate segment is mapped into the address space by providing a link in a code table to the starting address of the function. This link functions in the same manner as the claimed link stub, which in Applicant's Fig. 11, element 128, is shown to be a jump instruction to the relevant memory address. It is noted that Kempf states "the executing process linked with the program code segment transfers control to the starting address of the initialization function", while Applicant claims "generating a link stub for the symbol reference when the symbol reference is to an external location to access the external location [and] redirecting the instruction to the link stub". However, the discrepancies in language used therein does not preclude that Kempf achieves what is claimed.

9. Applicant argues on page 10, *"The Kempf reference provides no discussion of executing a jump instruction that refers to a link stub corresponding to an external location in a second domain. At best, the Kempf reference describes the well known process of directly linking program code and data through use of a symbol table, not executing a jump instruction that refers to a link stub. Nor does the Kempf reference describe executing a link stub, as no discussion of link stubs is present in the Kempf reference."*

Examiner respectfully disagrees. Primarily, Applicant's argument is directed to the allegation that Kempf does not disclose *"a jump instruction that refers to a link stub"*. However, this is not what is claimed. The claim language states that *"the link stub is a jump instruction to the external location"*, not vice versa (Claim 3). It is Examiner's position that Kempf discloses a link within a code table that refers to a jump instruction, as is claimed. Specifically, once a

Art Unit: 2127

program code segment or memory portion is mapped into the address space of the executing process, a link is generated within the code table of the address space. This link refers to the starting address of the appropriate code segment, which functionally achieves the same result as the claimed jump instruction. That is, a code segment outside the address space, i.e. protection domain, of an executing process can be executed by linking to the starting address of the code segment.

10. Applicant argues on page 11, *“The Kempf reference provides no discussion of ‘protection domains,’ much less at least one protection domain owning a portion of the system space. At best, the Kempf reference describes address space objects that allow execution in non-supervisor mode.”*

Examiner respectfully disagrees. While Kempf does not use the specific language of “protection domain”, the manner in which Kempf protects against unauthorized access or memory breaches by crossing address spaces is functionally the same as the claimed “protection domain.” Specifically, it is well known that a protection domain seeks to control memory access for all threads and processes within a single address space by ensuring that only threads and processes within that protection domain may have unlimited access to the resources and memory pages within that domain. Kempf achieves this protection with an address space manager that controls access to the memory regions, and mapping code segments across address spaces when the need arises, in the same manner as the claimed invention.

11. Applicant argues on page 11, *"Nowhere does Van Doorn describe a protection domain having a protection view that designates a set of protection domains for unrestricted memory access. At best, the Van Doorn reference describes protection domains each having a protection view of 'a set of physical memory pages to virtual mappings'...but not a protection view designating a set of protection domains for unrestricted memory access."*

Examiner respectfully disagrees. Specifically, the memory organization system of Van Doorn defines protection domains, wherein each protection domain has a view of a portion of the memory system for which it has unrestricted access, while system calls made to portions of memory outside the protection view must be passed through the Java Nucleus, which handles access rights across domains. Applicant incorrectly alleges that Van Doorn does not teach *"a protection view designating a set of protection domains for unrestricted memory access"*. Specifically, Van Doorn teaches exactly this sort of memory organization (col. 8 lines 45-51, "Each protection domain has, depending on its privileges, a view onto this address space. This view includes a set of physical memory pages to virtual mappings together with their corresponding access rights"). That is, each protection domain has unrestricted access to the memory pages within its protection view, but for memory pages outside that view, the Java Nucleus handles those calls and performs the appropriate mappings such that cross protection domains such that security breaches do not occur.

### ***Conclusion***

12. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

Art Unit: 2127

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

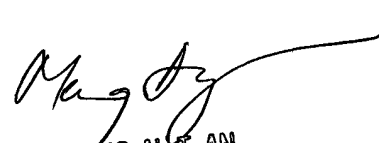
Any inquiry concerning this communication or earlier communications from the examiner should be directed to Syed J Ali whose telephone number is (703) 305-8106. The examiner can normally be reached on Mon-Fri 8-5:30, 1st Friday off.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, William A Grant can be reached on (703) 308-1108. The fax phone numbers for the organization where this application or proceeding is assigned are (703) 746-7239 for regular communications and (703) 746-7238 for After Final communications.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is (703) 305-3900.



Syed Ali  
February 25, 2004



MENG-AL T. AN  
SUPERVISORY PATENT EXAMINER  
TECHNOLOGY CENTER 2100